Accelerated Serverless Computing based on GPU Virtualization

Diana M. Naranjo*, Sebastián Risco*, Carlos de Alfonso*, Alfonso Pérez*, Ignacio Blanquer*, Germán Moltó*

[a]*Instituto de Instrumentación para Imagen Molecular (I3M)*
Centro mixto CSIC - Universitat Politècnica de València
Camino de Vera s/n, 46022, Valencia

Abstract

This paper introduces a platform to support serverless computing for scalable event-driven data processing that features a multi-level elasticity approach combined with virtualization of GPUs. The platform supports the execution of applications based on Docker containers in response to file uploads to a data storage in order to perform the data processing in parallel. This is managed by an elastic Kubernetes cluster whose size automatically grows and shrinks depending on the number of files to be processed. To accelerate the processing time of each file, several approaches involving virtualized access to GPUs, either locally or remote, have been evaluated. A use case that involves the inference based on deep learning techniques on transtoracic echocardiography imaging has been carried out to assess the benefits and limitations of the platform. The results indicate that the combination of serverless computing and GPU virtualization introduce an efficient and cost-effective event-driven accelerated computing approach that can be applied for a wide variety of scientific applications.

Keywords: Serverless Computing, GPUs, GPU Virtualization

*Corresponding author
Email addresses: `dnaranjo@i3m.upv.es` (Diana M. Naranjo), `serisgal@i3m.upv.es` (Sebastián Risco), `caralla@upv.es` (Carlos de Alfonso), `alpegon3@upv.es` (Alfonso Pérez), `iblanque@dsic.upv.es` (Ignacio Blanquer), `gmolto@dsic.upv.es` (Germán Moltó)

1. Introduction

Serverless computing [1] stands out as a computing paradigm that has been widely adopted by the industry for ultra-scalable event-driven processing on abstracted computational Clouds. Major public Cloud providers such as Amazon Web Services [2] have included services such as AWS Lambda [3] to support the definition and managed execution of functions. Those functions, coded in the supported programming languages, can be executed in response to certain events such as an HTTP request to an API Gateway [4] or a file upload to an object storage service, such as Amazon S3 [5]. AWS Lambda provides a convenient platform for processing a large number of short stateless independent jobs. To be more precise, up to 3000 parallel invocations is currently supported for up to 15 minutes of execution time using ephemeral storage and without the ability to perform communications among the invocations since no incoming TCP connections are supported.

Indeed, serverless computing has been widely adopted to support multiple use cases such as creating scalable web sites, real-time file processing, real-time stream processing, and Extract, Trasform and Load (ETL) processes, as indicated in the work by Lynn et al [6].

However, the following limitations of current serverless platforms offeblack by major public Cloud providers impose a serious restriction for their adoption in the scientific computing domain: i) limited maximum execution time, unfeasible for long-running scientific applications; ii) limited resources, since resource-intensive scientific applications may require beyond 3008 MB of RAM (current maximum memory size of AWS Lambda); iii) restricted execution environment, since scientific applications typically require a wide variety of libraries; iv) limited ephemeral storage, since 512 MB of disk space is insufficient to host the execution of applications with large dependencies and v) inability to access GPU resources within the function invocation for intensive workloads.

To this aim, a myriad of open-source serverless frameworks supporting the Functions as a Service (FaaS) [7] computing paradigm have surged in the last years, such as OpenFaaS [8] and Knative [9]. They execute functions, coded in certain programming languages, in response to HTTP events and other sources of events and typically rely on pre-provisioned computing platforms based on Container Orchestration Platforms (COP) such as Kubernetes [10] and Apache Mesos [11, 12]. These platforms are typically oriented to the execution of bursts of short HTTP-based requests, since they intend to mimic the functionality offeblack by their public Cloud providers counterparts.

However, serverless computing for data-processing scientific applications typically exhibit the following unique requirements: i) execution of long resource-intensive jobs; ii) ability to profit from accelerated computing supported by GPUs; iii) ability to support data-storage back-ends as sources of events and destination of data output and iv) ability to scale the underlying computing infrastructure to support incoming workloads without a major impact on the level of service.

We base our developments on our previous work in the field: OSCAR[1] [13], an open-source platform that builds on top of Kubernetes and OpenFaaS to support serverless computing for data-processing applications. Virtualization already plays a key role in this platform to isolate workloads and provide aggregated computing on top of the hardware. In this paper we investigate the integration of GPU virtualization into OSCAR in order to facilitate access to GPU computing for scientific workloads. Multiple approaches to GPU virtualization are assessed, from remotely accessing GPU devices using the rCUDA framework [14], to enabling direct access to the GPU devices via PCI passthrough by the Virtual Machine to be accessed by the container-based workloads being executed as a result of triggering a serverless function. Therefore, the main contribution to the state of the art of this paper is to analyse the integration of GPU computing and serverless computing through container-based workloads managed via Kubernetes.

After the introduction, the remainder of the paper is structublack as fol-

[1]OSCAR - `https://github.com/grycap/oscar`

lows. First, section 2 introduces the related work in the area and points out the unique features of the proposed platform. Second, section 3 describes the architecture of the proposed platform, its components and the integration of the GPU virtualization support. Third, section 4 assesses the benefits of the platform by means of a case study that integrates a deep learning application for transtoracic echocardiography imaging. Fourth, section 5 shows the results for the experiments performed comparing the use of CPUs, native GPUs and virtualized GPUs. Finally, section 6 summarises the main achievements of the paper and points to future work.

2. Related Work

This section describes the related work in the area of serverless computing, with a focus on the Functions as a Service (FaaS) model and, then, covers the state of the art of using GPUs in Cloud Computing. It is precisely at the intersection of these technologies that lies the main contribution of this work.

2.1. Serverless Computing and FaaS

Serverless computing has surged in the last years due, in part, to the dynamic allocation of computing resources managed by the Cloud provider together with the fine-grained pay-per-use billing model. Although some authors use the term interchangeably with FaaS, the latter stands for a computing model that allow application developers to execute functions without having to explicitly manage and scale the infrastructure needed to run such code. Public cloud providers such as AWS Lambda [3], Google Cloud Functions [15], Microsoft Azure Functions [16], Alibaba Cloud Function Compute [17], and IBM Cloud functions [18] offer the FaaS model as part as their serveless infrastructure package. Furthermore, to provide an on-premises open-source alternative to the offerings by the public Cloud providers, several frameworks that follow the FaaS model have appeablack in recent years. Some of the most widely used are: OpenFaaS [8], Knative [9], Fission [19], Nuclio [20], Apache OpenWhisk [21], Oracle Cloud Fn [22] and Riff [23], to name a few.

The work by Spillner et al. [24] presents the benefits of adopting the FaaS model for multiple scientific domains (e.g. computer graphics, cryptology, mathematics, and meteorology). However, due to some of the intrinsic characteristics of the FaaS model (e.g. pblackefined function environments, short-lived executions), there are not many scientific applications that can benefit from migrating to such model. Adapting established scientific applications to this new paradigm is not a trivial task and sometimes implies an application refactorization that it is outside of the knowledge scope of the developers of such applications. Nevertheless, there are works in the literature that successfully use this paradigm like the PyWren framework introduced by Jonas et al. [25]. The PyWren execution framework, in combination with LAmbdaPACK (i.e. a domain-specific language to implement linear algebra algorithms that are highly parallel), provides the base to the *numpywren* scientific computing framework by Shankar et al. [26]. Numpywren takes advantage of the function as a service model offeblack by AWS Lambda to solve linear algebra problems like large matrix multiplications and decomposition.

The SCAR[2] framework [27] also used AWS Lambda as a platform to execute general scientific applications based on Docker containers, to create highly-parallel event-driven file-processing serverless applications that execute on customized runtime environments provided by Docker images out of which containers are run on AWS Lambda. Another framework able to take advantage of AWS Lambda for intensive computing is MARLA[3] [28]. MARLA is able to execute Mapblackuce jobs as coordinated Lambda functions in response to file uploads to Amazon S3, achieving significant levels of performance without pre-provisioning infrastructure. These approaches can be applied to certain application use cases to achieve unprecedented levels of scalability (in the order of thousands of parallel invocations), when compablack to using virtual machines (in the order of tenths of virtual machines).

[2]SCAR - https://github.com/grycap/scar
[3]MARLA - https://github.com/grycap/marla

The research done by Kim et al. [29] also addresses the definition of a serverless platform with GPU support. However, it employs the NVIDIA-Docker runtime environment to allow function containers to have access to GPUs. This approach has the disadvantage that each GPU can only be accessed by one function invocation simultaneously. In contrast, our main contribution is the evolution of the OSCAR platform with rCUDA in order to allow the effective sharing of GPUs among functions to support a multitenant environment.

In addition, the rise of the artificial intelligence frameworks (e.g. TensorFlow [30], Microsoft Cognitive Toolkit [31] or Keras [32]) can profit from the usage of accelerated hardware (i.e. GPUs) not easily accessible from the function as a service architecture due to the many layers of virtualization employed. The work by Ishakian et al. [33] concluded that artificial intelligence workloads benefit from GPU support, specially for training neural networks. Thus, the inability to access GPU devices from existing serverless platforms may hinder its adoption for these workloads.

2.2. GPUs in the Cloud

Exploiting computing resources in the Cloud typically involves using Virtual Machines (VMs). While common hardware is well virtualized (i.e. CPU, Hard Disk, etc.), specific devices, especially those related with high performance, have not the same level of support and performance. This is the case of some network devices, FPGAs or GPUs. The work by Yu et al. [34] indicates that there are different approaches to exploit the physical GPUs using a VM: API remoting (or API forwarding), PCI-passthrough, and virtualizing the access to the GPU.

One of the first studies in this field was conducted by Lan Vu et al. [35] where a GPGPU virtualization solution named vmCUDA is proposed, which provides high-speed access of multiple virtual machines to shablack physical GPUs in VMware's ESX in order to offload general-purpose computing workloads.

In the case of API forwarding, rCUDA [14], vCUDA [36], GViM [37] and gVirtuS [38] are existing solutions that achieve good performance under certain circumstances. This approach is of special interest when the GPU resources are detached from the virtualization platform since it provides access to GPU devices from a virtualization platform that does not have direct access to them (i.e. the communication between the GPU and the VM is made using a network). Regarding the performance, the access to the GPU device is a competitive task between all the VMs that intend to use the GPUs, and the specific software is responsible for sharing the usage of the devices between the different client VMs.

The case of PCI passthrough has been used in production to obtain a high rate of efficiency in VMs, close to native performance using most of the hypervisors [39]. In the end, PCI passthrough is a technique that provides single VMs with exclusive access to a PCI device. While it obtains better performance, such exclusive and native access has several security implications that must be taken into account in a shablack virtualization platform (e.g. changing parameters of the physical device). Moreover, allocating one GPU for one VM results in an exclusive usage of that GPU, thus blackucing the efficiency.

The underlying idea in virtualizing the access to the GPU is to create virtual hardware that accesses the physical one, as it is done with the other components such as CPU or network devices. Examples of this approach are described in the work by Suzuki et al. [40] and Tan et al. [41]. Recently, vendors started providing support for virtualizing the access to the GPU. In particular, NVIDIA has introduced the vGPU support for some of its devices [42]. Using this approach, the access to the GPU is competitive such as when different processes access a physical GPU in a physical server, achieving near bare-metal performance [43].

This approach overcomes the problem of 1 to 1 allocation ratio that introduces the PCI-passthrough mechanism between VMs and GPUs. However, it may need additional software licenses to be used (e.g. NVIDIA Grid). Most public cloud providers (AWS, Microsoft Azure, Google Cloud Platform, Alibaba Cloud, etc.) support this kind of virtualization.

In order to address the open issue of supporting virtualized GPU-based computing in on-premises serverless computing frameworks, this paper presents the integration of GPU virtualization techniques into the open-source OSCAR platform in order to combine the benefits of the function as a service model with the high processing throughput offeblack by GPUs for the efficient execution of scientific applications.

3. Components and Architecture Design

This section introduces the main components and underlying technology employed before describing the general architecture of OSCAR with especial emphasis on the virtualization strategies of GPUs adopted.

3.1. rCUDA

rCUDA[4] is a framework developed at the Universitat Politècnica de València that allows the use of remote devices compatible with CUDA (Compute Unified Device Architecture) [44]. rCUDA creates virtual GPU devices which represent physical GPUs in remote machines that offer GPGPU (General Purpose Computing on Graphics Processing Units) services [45] for machines that do not have physically attached these devices.

The virtualization technique of GPUs represents a significant advantage in HPC clusters and datacenter environments since it increases the flexibility of the use of GPUs in the cluster. rCUDA allows sharing the same GPU among multiple applications, which encourages the development of a multitenant environment. The client/server architecture of rCUDA requires the server application to run on the machine where the GPUs are physically available. The clients use a library of wrappers to the CUDA Runtime API in order to access virtualized devices. Different communication protocols can be used in rCUDA such as Infiniband [46], RoCE (RDMA over Converged Ethernet) to optimize data exchange [47] and, finally, the TCP/IP protocol.

[4]rCUDA - `http://www.rcuda.net`

3.2. NVIDIA Container Runtime for Docker

NVIDIA-Docker[5] is a runtime developed by NVIDIA that allows the execution of Docker containers compatible with GPUs. This simplifies the usage of GPUs for applications that run inside a Docker container. NVIDIA-Docker includes driver-agnostic CUDA images and offers a wrapper that provides the containers with the necessary components to run the code on the GPU. The most recent evolution of this component is the NVIDIA Container Runtime, compatible with the Open Containers Initiative (OCI) specification [48].

3.3. Architecture

Figure 1 shows the architecture proposed to exploit remote GPU resources from an on-premises serverless computing platform.

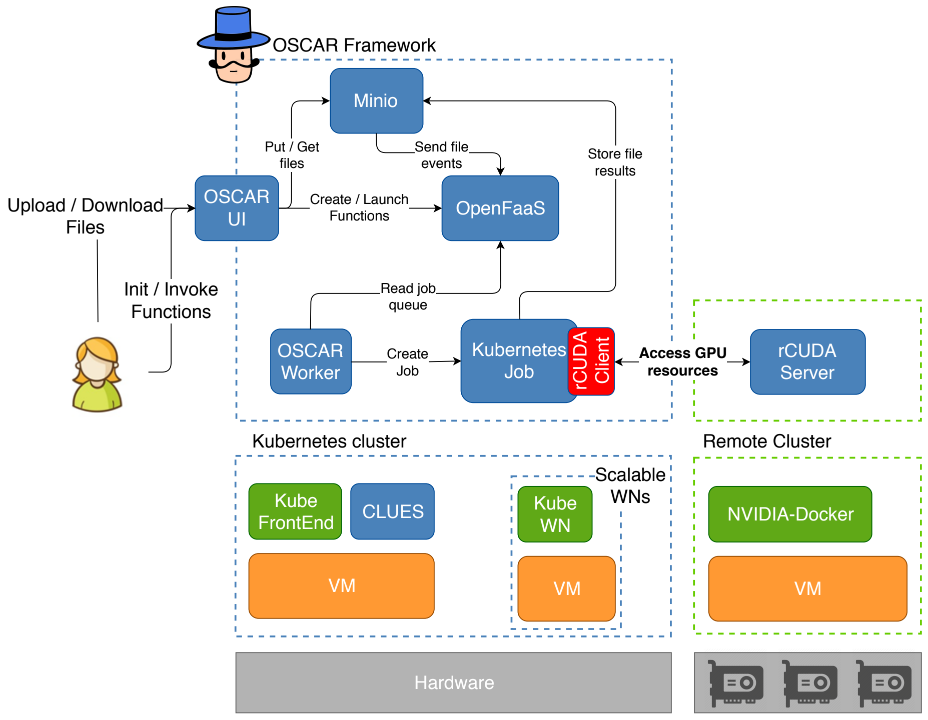

Figure 1: OSCAR architecture to access external GPU resources.

[5]NVIDIA-Docker - https://github.com/NVIDIA/nvidia-docker

The architecture is based on the OSCAR framework which allows application developers to execute Docker packaged applications as functions triggeblack in response to certain events, such as a file upload to a storage back-end.

The event-driven architecture that OSCAR offers simplifies the data and the infrastructure management abstracting all the configurations away from the end user. The application developers only need to define the execution environment (Docker image) and the script to be executed to perform the file processing in order to create a new function. Once the function is created, the invocation is triggeblack by uploading a file to the storage provider defined. For this, we use MinIO [49], an Amazon S3 compatible object storage system that provides both data persistence and the ability to trigger events. After the function invocation finishes, the results can be downloaded from the output storage provider used by the function. OSCAR has been designed to cope with long-running applications which makes it more suitable to execute scientific applications than traditional FaaS frameworks, which are typically oriented to processing bursts of HTTP-based stateless requests. For this, the function invocations are translated into Kubernetes jobs.

OSCAR services are deployed in an horizontally elastic Kubernetes cluster, whose nodes can be automatically added and removed on demand thanks to the CLUES[6] [50] elasticity manager and provisioned from multi-Clouds by means of the Infrastructure Manager (IM)[7] [51].

Figure 2 shows the integration of rCUDA in the architecture of OSCAR. In a remote cluster with physically attached GPUs runs the rCUDA server deployed inside a Docker container. For this it was necessary to use the NVIDIA-Docker runtime that allows virtualizing the GPUs inside the container where the rCUDA server is running. Notice that the rCUDA server can also be installed directly on the underlying operating system, instead of using Docker. The rCUDA client is run in the Kubernetes jobs and communication with the server

[6]CLUES - `https://github.com/grycap/clues`

[7]IM - `https://github.com/grycap/im`

is achieved via the network.

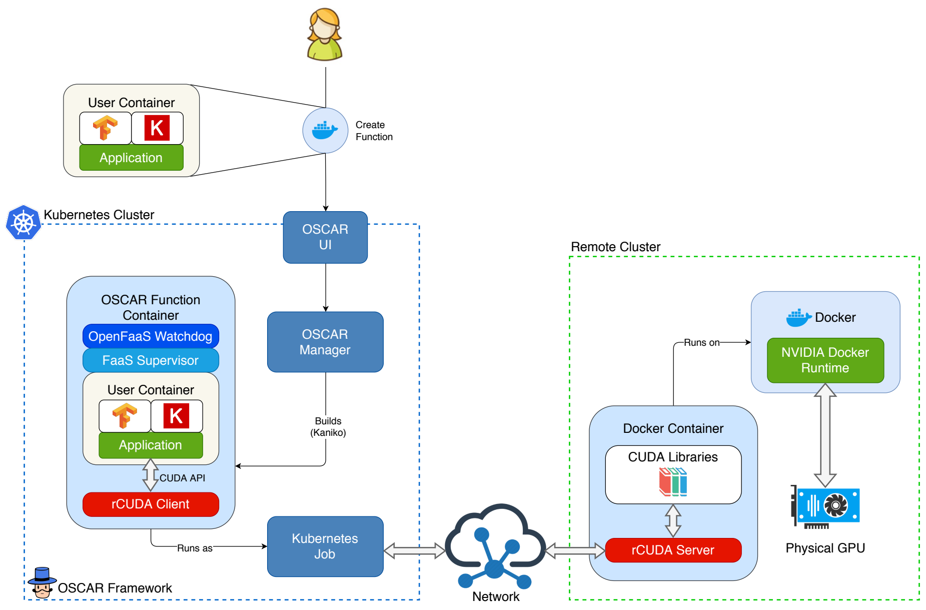

Figure 2: Integration of rCUDA into the OSCAR architecture.

Serverless functions are needed to be executed quickly and resources provisioned immediately. Therefore the integration of GPUs as back-end resources is key. However, GPU resources are expensive and may be shablack among multiple deployments for budget reasons. Usage of poweblack-on resources with GPUs should be maximized, so GPU virtualization comes into play. The article extends OSCAR with the support of mixed workloads that include standard CPU and GPU requirements. The integration of rCUDA brings addressing external network connectivity, overheads on loading additional dynamic libraries, resource contention and additional configuration issues at the level of OSCAR components.

While all the components used in the design of the architecture exist independently, the main contribution of this paper is to achieve their integration into a single open-source platform (OSCAR) that allows users to deploy serverless applications with seamless access to both native and virtualized GPUs.

4. Use case: Echocardiography classification on GPUs

In order to assess the benefits of the proposed platform, several scenarios have been designed to cover different GPU virtualization strategies. These scenarios have been applied to a real use case for the classification of echocardiography movies into pathological, borderline, and sound cases.

This classification use case applies an image classification model for echocardiography movies developed by QUIBIM[8] [52] [53] to classify an echocardiography movie frame according to the acquisition view into three categories (four-chamber, long-axis, short-axis). This classification is needed before applying other feature extraction technique, as they strongly depend on the acquisition view.

The echocardiography movies are obtained from the PROVAR study [54]. These movies have been obtained from a screening program where thousands of patients are exploblack yearly. Between 10 and 20 movies are acquiblack from each patient, using different techniques (morphological and Doppler) and from three different view angles. The acquiblack movie sequences last for around 2 to 4 seconds, comprising up to a few tens of frames. The size of each frame is 240 by 320 pixels.

All the frames in a movie are acquiblack from the same angle and modality. However, some frames are too noisy to infer the view. Therefore, it is important to extract all the frames, perform the classification of the individual frames and compute a consensus. The model has been developed using Keras[9] and leverages GPUs when available to speed up the training, validation and estimation phases. Figure 3 shows the processing pipeline.

In order to evaluate the best way to analyze the videos, different scenarios are proposed that include the execution of the classification code in CPUs and in local and remote GPUs. To compare such scenarios, 6 videos from the PROVAR study are used. Each video is divided into a number of frames determined

[8]QUIBIM - http://quibim.com/

[9]Keras - https://keras.io/

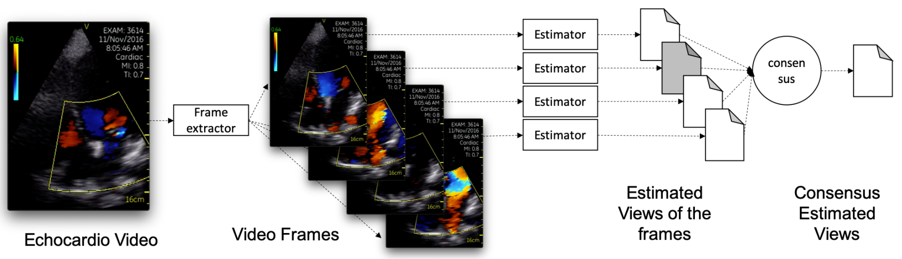

Figure 3: Processing pipeline for the use case.

according to the duration (60, 120, 180, 240, 300 and 360 frames), which allows us to test the performance when the workload increases.

In this particular use case, the model had already been previously trained. Therefore, we focused on classifying the segmented images of the videos according to the views. However, it is also possible to use the architecture in order to perform the training of the models using the GPUs, since it supports the execution of long-running jobs.

4.1. Requirements

Despite the fact that the most computationally intensive part is training the model using Deep Learning techniques, the cost of using the model for the classification, i.e. the inference phase, is not negligible. The possibility of parallelisation and the usage of accelerated devices can be the key to efficiently use the models in production. Indeed, the event-driven computing offeblack by serverless platforms is appropriate to compute the estimation of pblackictive models and the classification of complex data. The processing is triggeblack by uploading a set of movies corresponding to a specific patient which triggers the execution pipeline to perform the simultaneous processing of the files.

Therefore, the following requirements for this use case are defined:

- To exploit the inherent parallelism at the level of a video frame when processing the videos. Once extracted, individual frames can be processed concurrently. The developer of the application should not need to explicitly implement the parallel distribution of the frames along the processing nodes.

- To be able to use GPU acceleration from user-defined functions in the serverless platform. Functions should be able to transparently use GPU devices if they are available in the underlying computing node.
- To maximise the usage of GPU resources, potentially sharing them among different concurrent processes, in order to avoid locking a GPU device to a single CPU instance.
- To blackuce the gap between the development and the production environment, so applications validated in the development environment can be easily deployed in production with minimal risks.

These requirements come from a typical use case of unsupervised classification models. The creation of unsupervised classifiers require an intensive computing model building phase, where models are trained and validated, an error evaluation phase and, finally, the production phase for the use of the model to classify new data objects. In a scenario where data is continuously updated, models need to be retrained and, therefore, the use of GPU-enabled functions can ease the continuous training problem. If the complexity of the problem requires distributed training, other programming models should be exploblack. However, the scenario of using trained unsupervised classifiers for the estimation of the category of a new data, fits perfectly with the function as a service model. Since even the computing time of the estimation phase may is high, the use of GPUs appears to be highly convenient.

4.2. Computing Platform

The computing platform employed to execute the use case is composed of two nodes. The first one features two Skylake Gold 6130 at 2.1 GHz, 16 cores each, 768 GB RAM DDR4@2666, 10 GbE and includes an FPGA Arria 10 GX115 8GB, a RADEON Instinct MI25, 16GB, a Tesla P40 24GB and a Tesla V100 32GB. The second node features the same processor and memory together with 4 Tesla V100 32GB.

4.3. Scenarios

The following scenarios have been defined in order to evaluate different approaches to integrate GPU virtualization in an on-premises serverless platform:

1. Execution from Python console. The code that segments the video and classifies the images is executed from the Python console in a Docker container that has access to a single Tesla V100 GPU through NVIDIA-Docker. The main disadvantage of this approach is the manual configuration for each of the inputs and outputs of the code, far from being an automated process. This scenario determines the baseline execution time when accessing a GPU natively, to be more precise, using a lightweight virtualization approach.
2. OSCAR+CPU. This executes the image classification process in a container, through the OSCAR serverless platform, exclusively on the available CPU. This configuration allows to determine the improvement in the classification time of the image when moving from a CPU to a GPU.
3. OSCAR+Remote GPU (rCUDA). This approach provides serverless functions with access to remote GPUs from the containers of a Kubernetes cluster in which these functions are executed. The possibility of using these GPUs by multiple applications at the same time allows the implementation of a multi-tenant scenario in the use of GPUs. This is possible thanks to the functionality offeblack by rCUDA.
4. OSCAR+Native GPU. In this case the OSCAR platform was deployed on machines where the physical GPUs are available. To enable GPU support within the Kubernetes cluster it is first necessary to configure the working nodes with NVIDIA-Docker and then deploy the NVIDIA device plugin for Kubernetes[10] in the cluster. Once this is done, the NVIDIA GPUs can be consumed via container level resource requirements.

It is important to point out that all the scenarios that involve a GPU are

[10]NVIDIA device plugin for Kubernetes - https://github.com/NVIDIA/k8s-device-plugin

based on virtualization techniques. We discarded carrying out native executions on the physical nodes with GPU access since the platform is configublack as a multi-tenant on-premises Cloud and, therefore, users are not expected to run their code natively on these nodes.

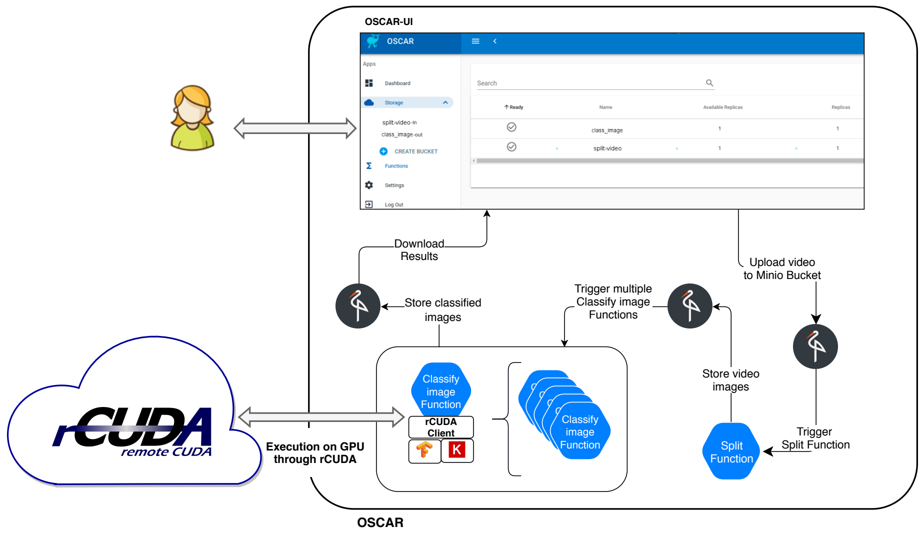

Figure 4: Workflow of the selected use case on the OSCAR platform using remote GPUs with two different functions.

Scenarios 2, 3 and 4 included two variants due to the fact that the code employed to classify an image needs to import the Tensorflow and Keras libraries. In the first variant, shown in Figure 4, the Keras and Tensorflow libraries are imported each time the segmentation function generates an image to be classified. This variant allows the execution of multiple classification functions in parallel, but increases the execution time by having to import the libraries when an image to be classified is generated. In the second variant (see Figure 5), these libraries are imported only once and then all the images from a single video are classified. A performance analysis of the processing time of both variants proved that performing all the image classifications in the same function was more efficient than carrying them out in separate functions. Therefore, the tests were made under these second variant. This means that a video will trigger the execution of a single function responsible for splitting it into images and performing the classification process for each image. Thus, parallelism can be achieved by processing multiple videos.

In scenario number three, where remote GPUs are used, this may have a significant impact since these libraries are imported through rCUDA and there is a non negligible time for the rCUDA server and client to establish this communication.

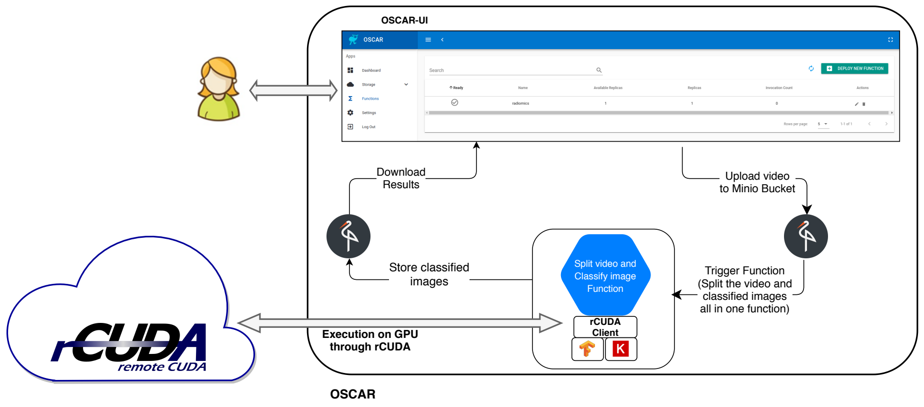
OSCAR-UI
Download Results
Upload video to Minio Bucket
Store classified images
Split video and Classify image Function
Trigger Function (Split the video and classified images all in one function)
rCUDA Client
Execution on GPU through rCUDA
rCUDA
remote CUDA
OSCAR

Figure 5: Workflow of the selected use case on the OSCAR platform using remote GPUs with all the code in one function.

In order to evaluate the elasticity of the architecture, the deployment of a new working node in the OSCAR cluster was triggeblack to compute the time taken to provision and configure it on the on-premises Cloud. This time was approximately 9 minutes, which can be significantly blackuced if pre-configublack Virtual Machine Images are used. It is important to point that this time was not taken into account when measuring the execution time of use cases since the focus of the paper is not on the elasticity of the platform but rather on the ability to introduce GPU support for the execution of the functions.

By defining these four scenarios we aim to compare the execution time of the image analysis resulting from the case study and, thus, determine the advantages and limitations for each scenario.

5. Results and Discussion

For each of the aforementioned scenarios, the processing time of the videos was measublack, taking into account the segmentation of the video into images and their classification using the pre-trained model based on Tensorflow and Keras.

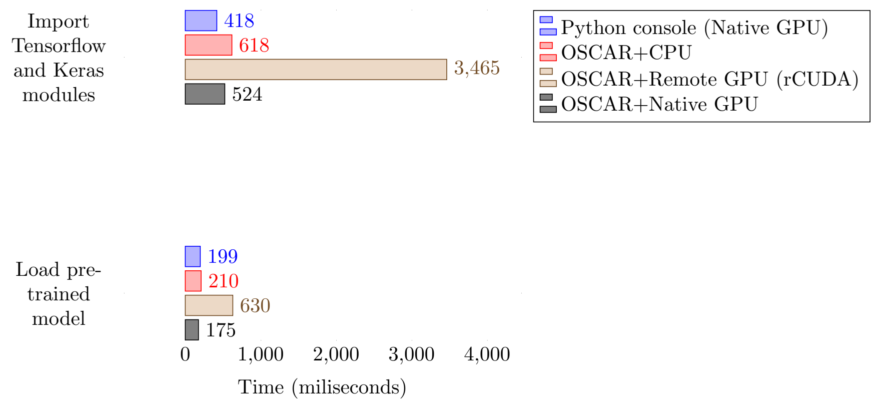

Figure 6: Average time importing the requiblack modules and loading the pre-trained model in the analysed scenarios.

Figure 6 shows the average time it takes to load the training model and the Tensorflow and Keras libraries. As can be seen in the graph, when virtualizing the GPUs within the cluster through rCUDA the time increases considerably since libraries need to be loaded by the rCUDA server. Indeed, this component runs in a virtual machine where it has access to a Tesla V100 32GB GPU. The rCUDA client runs in the Kubernetes jobs on a virtual machine in which GPUs are not accessible. The connection between the client and the server is through the 10GbE network card of the physical machine. It was precisely in order to minimize this overhead that we decided to adopt variant 2, i.e. have a single function invocation to process a single video, in all the subsequent tests.

Several executions were carried out to process a set of videos using the on-premises Cloud platform and a negligible time difference was observed among them. Therefore, Figure 7 show the total processing time including the segmen-

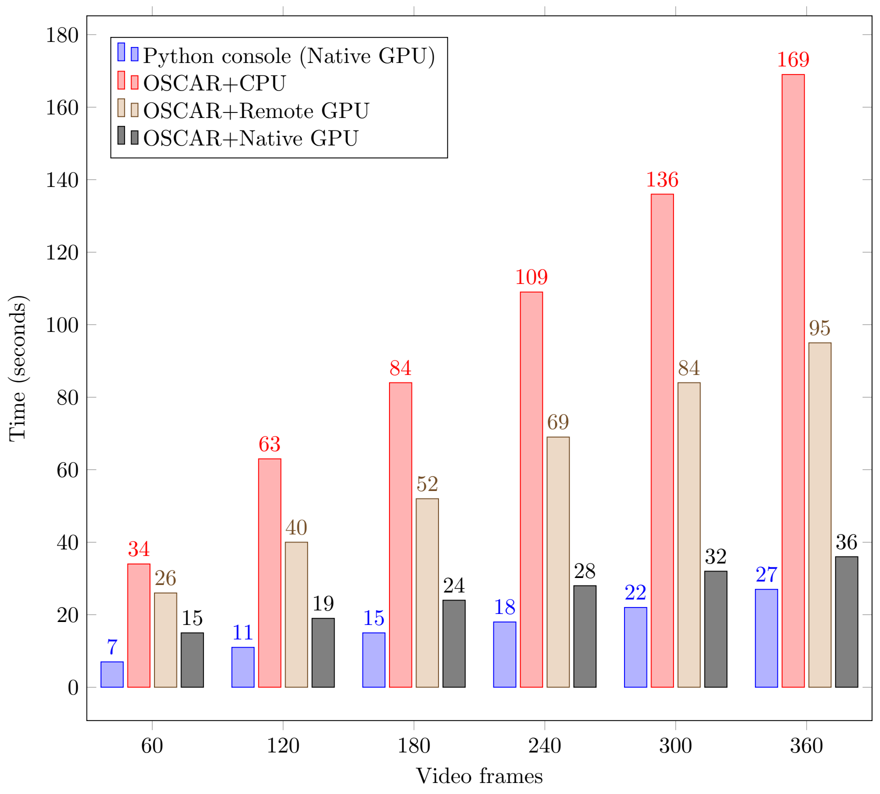

Figure 7: Comparison classifying different length videos in the analysed scenarios.

tation time of the video into images, loading the libraries (as shown in Figure 5) and the classification of the images. The graphs reveal that the use of the CPU results is the slowest procedure, compablack to using GPUs, especially in cases where the videos generate a large number of frames.

Concerning the usage of GPUs, scenario 1, i.e. Python console (Native GPU), which provides lightweight access to a Tesla V100 GPU via NVIDIA-Docker results, is the fastest approach. The disadvantage of this approach is that it is necessary to manually run the code for each of the videos, or via scripts, and does not have the possibilities offeblack by a serverless platform in terms of parallel processing of multiple executions coupled with automated elasticity based on the numbers of virtual machines of the Kubernetes cluster.

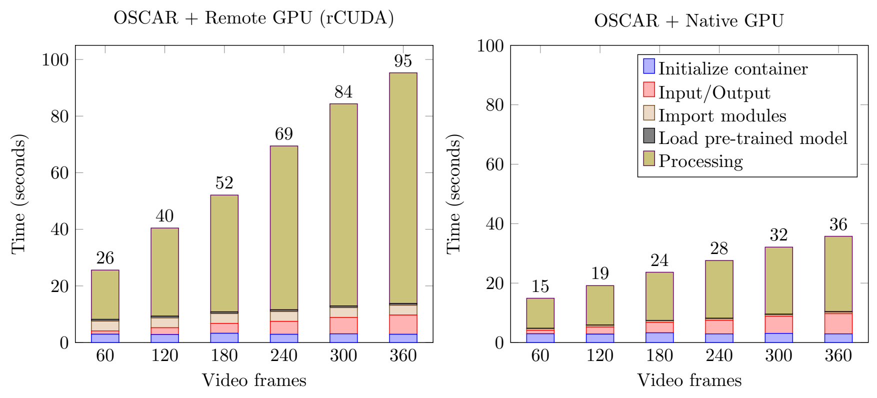

Figure 8: Time segmentation according to the main execution phases in scenarios 3 and 4.

With the use of CPUs (scenario 2), image processing times are higher than with the use of GPUs, mainly in videos that are divided into more frames what increases the computing requirements.

In the case of scenarios 1 and 4, where GPUs are used natively, a difference in execution time is noticed. The execution time using the Python console is less than when using OSCAR + Native GPU. The reason is that when using OSCAR the classification code is executed through the invocation of the function, starting from the fragmentation of the video into images, until their classification. However, in the case of the Python console, the classification code is executed manually. Hence, the fundamental execution time difference lies in the creation of the environment for the execution of the functions in the OSCAR platform which, of course, does not appear when using the Python console.

Scenario 3 (OSCAR + Remote GPU) virtualizes the access to remote GPUs, via rCUDA, from the functions executed inside the Docker containers that are run in the Kubernetes cluster. This enables multiple classification functions to be executed while simultaneously accessing the same GPU, enabling the development of a multi-tenant environment.

With the results obtained in scenario 4 (OSCAR + Native GPU), it is evident that with the use of native GPUs in a Kubernetes cluster the processing is faster since the access to the GPUs is local. Figure 8 shows the time distribution among the main phases requiblack for the execution of the function. The *Initialize container* and *Input/Output* phases are executed by the OSCAR framework and depend on the start-up of the execution environment and the upload and download time of the input and output files. Therefore, the larger the video to process, the longer it takes.

In the OSCAR + Remote GPU (rCUDA) scenario the processing is significantly slower than when using native GPUs, since rCUDA is a client-server application that makes extensive use of the network. Consequently , it is recommended to use InfiniBand networks to achieve better performance. In this case, the phases *Import modules*, *Load pre-trained model* and *Processing* are due to the GPU virtualization overhead and depend fundamentally on the network speed.

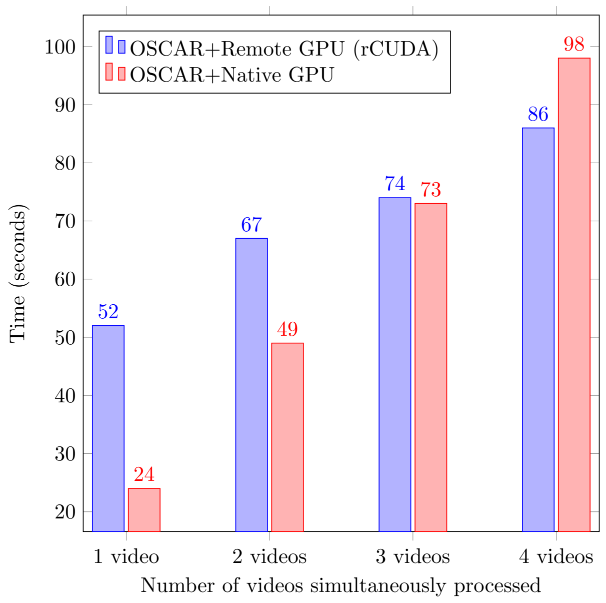

Figure 9: Execution times for processing up to 4 videos (180 frames) in scenario 3 and 4.

When using native GPUs, the NVIDIA complement for Kubernetes does not currently allow simultaneous access to GPU resources. To this aim, the use of rCUDA allows the development of a multitenant environment where several applications simultaneously access the same GPU, thus obtaining better performance.

Figure 9 shows the execution times of up to 4 videos processed simultaneously in the cases of OSCAR + Remote GPU and OSCAR + Native GPU. It is observed that as the workload increases, in the case of using a native GPU a bottleneck is created due to the NVIDIA Kubernetes plugin that only allows the use of one GPU per pod. In the case of rCUDA, when the amount of videos to be processed increases, better results are obtained due to the possibility of sharing the GPU between applications simultaneously.

6. Conclusions and Future Work

This paper has focused on the integration of GPU virtualization techniques in on-premises serverless computing platforms based on containers. In particular, we addressed both remote GPU virtualization, through rCUDA, and lightweight virtualization through NVIDIA-Docker.

Based on the OSCAR serverless platform, which provides event-driven computing for function invocations that are run as jobs in a Kubernetes cluster, we assessed the integration of both GPUs and serverless computing to provide accelerated support for functions.

A use case on transtoracic echocardiography imaging that uses machine learning techniques to perform segmentation and classification of images was integrated to achieve video processing on dynamically deployed Kubernetes clusters in an on-premises Clouds with both GPU-enabled and CPU-only nodes.

The results indicated that function reorganization in order to minimize the loading time of deep learning libraries is important to minimize the execution time. Direct access to GPUs provides the most efficient approach. Using rCUDA, which provides GPU virtualization over the network achieved good performance when compablack to using CPUs but, obviously, not close to the one achieved with direct access. Still, rCUDA's ability to support multi-tenant access to a single GPU revealed to be a powerful capability in order to support enhanced parallelism for serverless platforms. The integration of rCUDA into the OSCAR architecture constitutes an important advance in serverless computing and in the shablack access of a GPU by multiple applications in a Kubernetes cluster.

Future work involves the integration of a plugin of Kubernetes with the aim of eliminating the bottleneck introduced in the processing of several videos simultaneously and thus sharing a GPU among multiple pods. In addition, the integration of OSCAR with SCAR is intended to achieve hybrid workloads based in on-premises and public Clouds.

7. Acknowledgement

The work presented in this article has been partially funded by a research grant from the regional government of the Comunitat Valenciana (Spain), co-funded by the European Union ERDF funds (European Regional Development Fund) of the Comunitat Valenciana 2014-2020, with reference IDIFEDER/2018/032 (High-Performance Algorithms for the Modelling, Simulation and early Detection of diseases in Personalized Medicine). The authors would also like to thank the Spanish "Ministerio de Economa, Industria y Competitividad" for the project "BigCLOE" with reference number TIN2016-79951-R and the project ATMOSPHERE, funded jointly by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement No 777154 and the Brazilian Ministério de Ciência, Tecnologia e Inovação (MCTI), number 51119. D.M.N would like to thank the "Generalitat Valenciana" for the grant GrisolíaP/2017/071.

References

[1] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. M. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. A. Popa, D. A. Patterson, J. Carreira, J. E. Gonzalez, Cloud

Programming Simplified: A Berkeley View on Serverless Computing, Tech. rep. (feb 2019). arXiv:1902.03383, doi:arXiv:1902.03383v1.
URL http://arxiv.org/abs/1902.03383http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html

[2] Amazon, Amazon Web Services (AWS).
URL http://aws.amazon.com

[3] Amazon Web Services, AWS Lambda.
URL https://aws.amazon.com/lambda

[4] A. W. Services, API Gateway.
URL https://aws.amazon.com/api-gateway

[5] Amazon, Amazon Simple Storage Service (Amazon S3).
URL http://aws.amazon.com/s3/

[6] T. Lynn, P. Rosati, A. Lejeune, V. Emeakaroha, A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms, in: Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, Vol. 2017-Decem, IEEE, 2017, pp. 162–169. doi:10.1109/CloudCom.2017.15.
URL http://ieeexplore.ieee.org/document/8241104/

[7] E. van Eyk, A. Iosup, S. Seif, M. Thömmes, The SPEC cloud group's research vision on FaaS and serverless architectures, Proceedings of the 2nd International Workshop on Serverless Computing - WoSC '17 (2017) 1–4doi:10.1145/3154847.3154848.
URL http://dl.acm.org/citation.cfm?doid=3154847.3154848

[8] A. Ellis, OpenFaaS.
URL https://www.openfaas.com/

[9] Google, Knative.
URL https://github.com/knative/

[10] Kubernetes, Kubernetes.
URL https://kubernetes.io/

[11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, I. Stoica, Mesos: a platform for fine-grained resource sharing in the data center (2011) 295–308.
URL http://dl.acm.org/citation.cfm?id=1972457.1972488

[12] Apache Mesos.
URL http://mesos.apache.org/

[13] A. Pérez, S. Risco, D. M. Naranjo, M. Caballer, G. Moltó, Serverless Computing for Event-Driven Data Processing Applications, in: 2019 IEEE International Conference on Cloud Computing (CLOUD 2019), 2019.

[14] J. Duato, A. J. Pena, F. Silla, R. Mayo, E. S. Quintana-Orti, rCUDA: Reducing the number of GPU-based accelerators in high performance clusters, in: 2010 International Conference on High Performance Computing & Simulation, IEEE, 2010, pp. 224–231. doi:10.1109/HPCS.2010.5547126.
URL http://ieeexplore.ieee.org/document/5547126/

[15] Google, Google Cloud Functions.
URL https://cloud.google.com/functions/

[16] Microsoft, Microsoft Azure Functions.
URL https://azure.microsoft.com/en-us/services/functions/

[17] Alibaba, Alibaba Cloud Function Compute.
URL https://www.alibabacloud.com/products/function-compute

[18] IBM, IBM Cloud Functions.
URL https://www.ibm.com/cloud/functions

[19] Fission.
URL https://fission.io/

[20] Nuclio.
URL https://nuclio.io/

[21] Apache, OpenWhisk.
URL https://openwhisk.apache.org/

[22] Oracle, Fn Project.
URL https://fnproject.io/

[23] Pivotal, Project riff.
URL https://projectriff.io/

[24] J. Spillner, C. Mateos, D. A. Monge, Faaster, better, cheaper: the prospect of serverless scientific computing and HPC, in: Communications in Computer and Information Science, Vol. 796, Springer, Cham, 2018, pp. 154–168. doi:10.1007/978-3-319-73353-1_11.
URL http://link.springer.com/10.1007/978-3-319-73353-1{_}11

[25] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, B. Recht, Occupy the cloud: distributed computing for the 99%, in: Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17, ACM Press, New York, New York, USA, 2017, pp. 445–451. arXiv:1702.04024, doi:10.1145/3127479.3128601.
URL http://dl.acm.org/citation.cfm?doid=3127479.3128601

[26] V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht, J. Ragan-Kelley, numpywren: serverless linear algebraarXiv:1810.09679.
URL https://arxiv.org/abs/1810.09679

[27] A. Pérez, G. Moltó, M. Caballer, A. Calatrava, Serverless computing for container-based architectures, Future Generation Computer Systems 83 (2018) 50–59. doi:10.1016/j.future.2018.01.022.
URL http://linkinghub.elsevier.com/retrieve/pii/S0167739X17316485

[28] V. Giménez-Alventosa, G. Moltó, M. Caballer, A framework and a performance assessment for serverless MapReduce on AWS Lambda, Future Generation Computer Systemsdoi:10.1016/j.future.2019.02.057.
URL https://linkinghub.elsevier.com/retrieve/pii/S0167739X18325172

[29] J. Kim, T. J. Jun, D. Kang, D. Kim, D. Kim, Gpu enabled serverless computing framework, in: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), 2018, pp. 533–540. doi:10.1109/PDP2018.2018.00090.

[30] Google, Tensorflow.
URL https://www.tensorflow.org/

[31] Microsoft, Microsoft Cognitive Toolkit.
URL https://www.microsoft.com/en-us/cognitive-toolkit/

[32] Keras, Keras.
URL https://keras.io/

[33] V. Ishakian, V. Muthusamy, A. Slominski, Serving deep learning models in a serverless platform, in: 2018 IEEE International Conference on Cloud Engineering (IC2E), 2018, pp. 257–262. doi:10.1109/IC2E.2018.00052.

[34] H. Yu, C. J. Rossbach, Full virtualization for gpus reconsidered, 2017.

[35] L. Vu, H. Sivaraman, R. Bidarkar, Gpu virtualization for high performance general purpose computing on the esx hypervisor, Vol. 46, 2014.

[36] L. Shi, H. Chen, J. Sun, K. Li, vcuda: Gpu-accelerated high-performance computing in virtual machines, IEEE Transactions on Computers 61 (6) (2012) 804–816. doi:10.1109/TC.2011.112.

[37] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, P. Ranganathan, Gvim: Gpu-accelerated virtual machines, in: Proceedings

of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing, HPCVirt '09, ACM, New York, NY, USA, 2009, pp. 17–24. doi:10.1145/1519138.1519141.
URL http://doi.acm.org/10.1145/1519138.1519141

[38] G. Giunta, R. Montella, G. Agrillo, G. Coviello, A gpgpu transparent virtualization component for high performance computing clouds, in: P. D'Ambra, M. Guarracino, D. Talia (Eds.), Euro-Par 2010 - Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 379–391.

[39] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, G. C. Fox, Gpu passthrough performance: A comparison of kvm, xen, vmware esxi, and lxc for cuda and opencl applications, in: 2014 IEEE 7th International Conference on Cloud Computing, 2014, pp. 636–643. doi:10.1109/CLOUD.2014.90.

[40] Y. Suzuki, S. Kato, H. Yamada, K. Kono, Gpuvm: Gpu virtualization at the hypervisor, IEEE Transactions on Computers 65 (9) (2016) 2752–2766. doi:10.1109/TC.2015.2506582.

[41] H. Tan, Y. Tan, X. He, K. Li, K. Li, A virtual multi-channel gpu fair scheduling method for virtual machines, IEEE Transactions on Parallel and Distributed Systems 30 (2) (2019) 257–270. doi:10.1109/TPDS.2018.2865341.

[42] NVIDIA, What is a virtual gpu (2018).
URL https://blogs.nvidia.com/blog/2018/06/11/what-is-a-virtual-gpu/

[43] U. Kurkure, Episode 3: Performance comparison of native gpu to virtualized gpu and scalability of virtualized gpus for machine learning (2017).
URL https://blogs.vmware.com/performance/2017/10/episode-3-performance-comparison-native-gpu-virtualized-gpu-scalability-virtualized-gpus-machine-learning.html

[44] M. Ujaldón, CUDA achievements and GPU challenges ahead, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9756, 2016, pp. 207–217. doi:10.1007/978-3-319-41778-3_20.
URL https://link.springer.com/chapter/10.1007/978-3-319-41778-3{_}20

[45] C. Reano, F. Silla, A Performance Comparison of CUDA Remote GPU Virtualization Frameworks, in: 2015 IEEE International Conference on Cluster Computing, IEEE, 2015, pp. 488–489. doi:10.1109/CLUSTER.2015.76.
URL http://ieeexplore.ieee.org/document/7307623/

[46] F. Pérez, C. Reaño, F. Silla, Providing CUDA acceleration to KVM virtual machines in InfiniBand clusters with rCUDA, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9687, 2016, pp. 82–95. doi:10.1007/978-3-319-39577-7_7.
URL https://link.springer.com/chapter/10.1007/978-3-319-39577-7{_}7

[47] C. Reaño, F. Silla, G. Shainer, S. Schultz, Local and Remote GPUs Perform Similar with EDR 100G InfiniBand, in: Proceedings of the Industrial Track of the 16th International Middleware Conference on ZZZ - Middleware Industry '15, ACM Press, New York, New York, USA, 2015, pp. 1–7. doi:10.1145/2830013.2830015.
URL http://dl.acm.org/citation.cfm?doid=2830013.2830015http://dx.doi.org/10.1145/2830013.2830015

[48] Open Container Initiative.
URL https://www.opencontainers.org/

[49] MinIO.
URL https://min.io

[50] C. de Alfonso, M. Caballer, F. Alvarruiz, V. Hernández, An energy management system for cluster infrastructures, Computers & Electrical Engineering 39 (8) (2013) 2579–2590.
URL http://www.sciencedirect.com/science/article/pii/S0045790613001365

[51] M. Caballer, I. Blanquer, G. Moltó, C. de Alfonso, Dynamic Management of Virtual Infrastructures, Journal of Grid Computing 13 (1) (2015) 53–70. doi:10.1007/s10723-014-9296-5.
URL http://link.springer.com/article/10.1007/s10723-014-9296-5http://link.springer.com/10.1007/s10723-014-9296-5

[52] A. Jimenez-Pastor, A. Alberich-Bayarri, F. Garcia-Castro, L. Marti-Bonmati, Automatic visceral fat characterisation on ct scans through deep learning and cnn for the assessment of metabolic syndrome., in: ECR 2019: Book of Abstracts. Insights into Imaging., Vol. 10(S1), 2019.

[53] E. Camacho-Ramos, A. Jimenez-Pastor, I. Blanquer, F. Garca-Castro, A. Alberich-Bayarri, Computer aided diagnosis for Rheumatic Heart Disease by AI applied to features extraction from echocardiography.

[54] B. R. Nascimento, A. Z. Beaton, M. C. P. Nunes, A. C. Diamantino, G. A. Carmo, K. K. Oliveira, C. M. Oliveira, Z. M. A. Meira, S. R. T. Castilho, E. L. Lopes, I. M. Castro, V. M. Rezende, G. Chequer, T. Landay, A. Tompsett, A. L. P. Ribeiro, C. Sable, Echocardiographic prevalence of rheumatic heart disease in brazilian schoolchildren: Data from the provar study, International Journal of Cardiology 219 (2016) 439 – 445. doi:https://doi.org/10.1016/j.ijcard.2016.06.088.
URL http://www.sciencedirect.com/science/article/pii/S0167527316310907